ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (011) 818-5583 Fax (011) 818-5294

Departamento de Engenharia de Computação e Sistemas Digitais

Segunda Avaliação de Linguagens e Compiladores

Felipe Giunte Yoshida - N°4978231

1° de Dezembro de 2009

A linguagem LazyComb é minimalista, funcional, com coletor de lixo (*garbage collector*), com transparência referencial e com um sistema de E/S simples baseado em *stream*. A linguagem captura a essência da programação funcional e seus programas são definidos por meio do uso de **combinadores**. Um combinador executa uma ação que pode ser implementada através de uma abstração lambda. Uma abstração lambda é uma especificação de função anônima com parâmetros como, por exemplo, a definição da função identidade, $(lambda(x)x)$, assim a aplicação de um valor à identidade retornará o próprio valor: $(lambda(x)x)3 \rightarrow 3$. O combinador I tem exatamente essa função: $I3 \rightarrow 3$. Entretanto, não há valores numéricos, eles devem ser todos representados como funções, assim o numeral 0 é representado por $(lambda(fx)(fx))$, logo 1 é representado por $(lambda(fx)(f(fx)))$. Desta maneira qualquer número natural é representado pela composição funcional. Usando combinadores o número 256 fica **SII(SII(S(S(KS)K)I))**.

Sintaxe		Semântica
`Program`	`::= CCExpr`	`CCExpr`
`CCExpr`	`::= CCExpr Expr`	`(CCExpr Expr)`
	`| epsilon`	`(lambda (x) x)`
`Expr`	`::= "i"`	`(lambda (x) x)`
	`| Expr'`	`Expr'`
`IotaExpr`	`::= "i"`	`(lambda (x) (x S K))`
	`| Expr'`	`Expr'`
`Expr'`	`::= "I"`	`(lambda (x) x)`

```
            | "K" | "k"                 (lambda (x y) x)
            | "S" | "s"                 (lambda (x y z) ((x z) (y z)))
            | NonemptyJotExpr           NonemptyJotExpr
            | "‘" Expr1 Expr2           (Expr1 Expr2)
            | "*" IotaExpr1 IotaExpr2   (IotaExpr1 IotaExpr2)
            | "(" CCExpr ")"            CCExpr

  NonemptyJotExpr
           ::= JotExpr "0"              (JotExpr S K)
             | JotExpr "1"              (lambda (x y) (JotExpr (x y)))

  JotExpr  ::= NonemptyJotExpr          NonemptyJotExpr
             | epsilon                  (lambda (x) x)
```

Considerando que a execução de um programa em LazyComb é baseada na aplicação de combinadores a expressões, ou seja, cada programa será executado a partir de uma lista prefixada de operadores que terminará, pede-se:

Construa um reconhecedor determinístico, baseado no autômato de pilha estruturado, que aceite como entrada válida um programa em LazyComb.

1 Linguagem

Convertendo a gramática descrita acima para a notação de Wirth:

```
Program = CCExpr .

CCExpr =  { Expr } .

Expr = "i" | Expr' .

IotaExpr = "i" | Expr' .

Expr' =  "I" | "K" | "S" | "k" | "s" | NonemptyJotExpr | "‘" Expr Expr
| "*" IotaExpr IotaExpr | "(" CCExpr ")" .

NonemptyJotExpr = JotExpr ( "0" | "1" ) .

JotExpr = { NonemptyJotExpr } .
```

Simplificando, chegamos a:

```
Program = { Expr } .
```

```
Expr = "i" | "I" | "K" | "S" | "k" | "s" | ( "0" | "1" ) { ( "0" | "1" ) }
| "`" Expr Expr | "*" Expr Expr | "(" { Expr } ")" .
```

2 Análise léxica

O analisador léxico da linguagem é extremamente simples e pode ser observado abaixo.

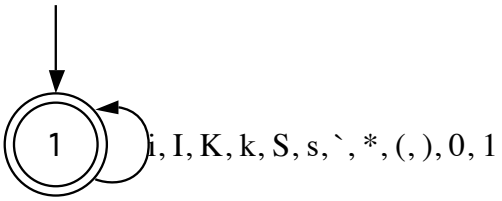

Figura 1: Analisador léxico

Sua simplicidade se dá porque todos os tokens são constituídos de apenas um caractere. Além disso, todos eles são previamente definidos.

3 Análise sintática

Submáquina *Program*:
$Program = \bullet_0\{\bullet_1 Expr \bullet_2\}\ \bullet_1$.

Submáquina *expr*:
$Expr = \bullet_0 \text{“}i\text{”}\ \bullet_1 \mid \bullet_0\ \text{“}I\text{”}\ \bullet_2 \mid \bullet_0\ \text{“}K\text{”}\ \bullet_3 \mid \bullet_0\ \text{“}S\text{”}\ \bullet_4 \mid \bullet_0\ \text{“}k\text{”}\ \bullet_5 \mid \bullet_0\ \text{“}s\text{”}\ \bullet_6 \mid \bullet_0\ (\bullet_0 \text{“}0\text{”}\ \bullet_8 \mid \bullet_0\ \text{“}1\text{”}\bullet_9)\ \bullet_7\ \{\bullet_{10}(\bullet_{10} \text{“}0\text{”}\ \bullet_{11} \mid \bullet_{10}\ \text{“}1\text{”}\bullet_{12})\}\ \bullet_{10} \mid \bullet_0\ \text{“`”}\ \bullet_{13}\ Expr\ \bullet_{14}\ Expr\ \bullet_{15} \mid \bullet_0\ \text{“}*\text{”}\ \bullet_{16} Expr\ \bullet_{17}\ Expr\ \bullet_{18} \mid \bullet_0\ \text{“(”}\bullet_{19}\{\bullet_{20} Expr \bullet_{21}\}\ \bullet_{20}\ \text{“)”}\ \bullet_{22}$.

Eliminando o não-determinismo e retirando os estados inválidos, chegamos aos autômatos presentes nas figuras 2 e 3.

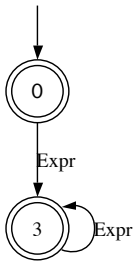

Figura 2: Autômato finito determinístico de *program*

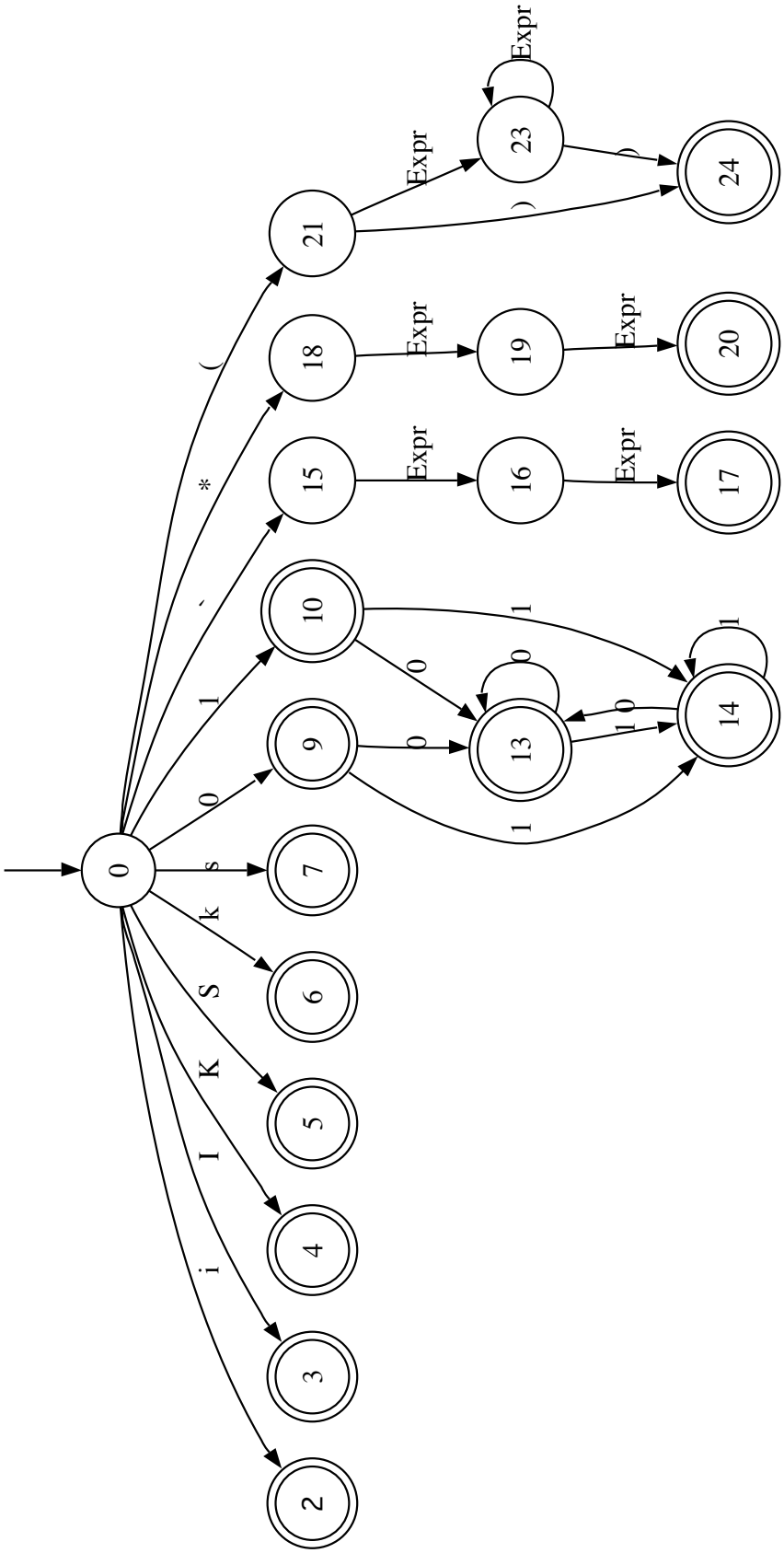

Figura 3: Autômato finito determinístico de *expr*